



Fundamentos de Big Data

Lección 2: Librerías para Gráficas

ÍNDICE

Lección 2. – Librerías para Gráficas.....	2
Presentación y objetivos	2
1. Gráficas en Data Science - ¿ Qué debo utilizar ?	3
2. Obtención de Información concreta en Dataframes	4
3. Gráficos rápidos con Pandas para Dataframes	7
4. Gráficas del Iris Dataset – Para el Pétalo	16
5. Gráficas del Iris Dataset – Para el Sépalo.....	20
6. Otras Librerías para hacer Gráficos.....	23
7. Herramientas Avanzadas para hacer gráficos.....	24
8. Herramientas Avanzadas 1: Dash.....	25
9. Herramientas Avanzadas 2: Bokeh	26
10. Herramientas Avanzadas 3: Power BI.....	27
11. Herramientas Avanzadas 4: Tableau.....	28
12. Herramientas Avanzadas 5: Savvy	30
13. Herramientas Avanzadas 6: Folium	31
14. Puntos clave	32

Lección 2. – Librerías para Gráficas

PRESENTACIÓN Y OBJETIVOS

Esta asignatura contiene una gran cantidad de elementos teóricos, que han sido sintetizados para tratar de hacer cosas prácticas.

Antes de explicar algunos posibles Frameworks con los que se puede trabajar para hacer gráficos trataremos de explicar lo que es el Data Mining y el análisis de datos exploratorio con un Dataset típico al comenzar en Data Science, el Titanic Dataset.

En el caso que nos ocupa daremos una visión amplia de las posibles librerías que tenemos para hacer gráficos en Data Science, algunas ya han sido explicadas con anterioridad.

Respecto a los Frameworks que hablaremos en temas sucesivos, por cierto, ya fue explicado Dash, en el tema de Aplicaciones con Python y volveremos al mismo, para explicar más cosas del mismo.



Objetivos

- Recordar conceptos básicos de gráficos con pandas, Matplotlib, Seaborn, etc. (Tomaremos como punto de partida el Iris Dataset ya visto)

1. GRÁFICAS EN DATA SCIENCE - ¿ QUÉ DEBO UTILIZAR ?

Lo Ideal para comenzar en este tema es recordar los conocimientos que teníamos de gráficas, explicados en Creación de Aplicaciones Python.

Para ello, y en esta ocasión trataremos de ver rápidamente las ventajas que aportan librerías como Seaborn, ploteando gráficas de gran calidad en poco tiempo.

Para ello, pues, haremos esta comparativa: Pandas-Matplotlib-Seaborn

(Aunque hay que comentar que existen muchas más opciones).

Y, en este caso, veremos cuando usar Pandas, cuando usar Matplotlib y cuando Seaborn, como deducción de los propios ejemplos.

En conclusión, y para que se entienda, ya se puede adelantar previamente que “pandas” es un tipo de gráficas que usaremos en contadas ocasiones, y simplemente cuando queramos obtener un gráfico muy rápido de un dataframe (df) añadiendo .plot() al dataframe (df), por ejemplo, un gráfico de barras, que ya veremos cómo se hace eso. Pero por lo general usaremos Seaborn cuando necesitemos algo más elaborado. Matplotlib tiene sus ventajas conocerlo, y lo usamos algunas veces.

Para esta comparativa, estaremos usando Jupyter Notebook como Entorno de Desarrollo.

Lo que presentamos a continuación son los pantallazos del código que necesitamos así como las propias gráficas que obtenemos.

Comencemos!

2. OBTENCIÓN DE INFORMACIÓN CONCRETA EN DATAFRAMES

PRINCIPALES DEPENDENCIAS

```
In [1]: import pandas as pd
```

Creo un DataFrame para el ejemplo

```
In [2]: df = pd.DataFrame({"X": [10,20,30,40,50], "y": [1,1,0,1,0]})  
df.head()
```

Out[2]:

	X	y
0	10	1
1	20	1
2	30	0
3	40	1
4	50	0

Figura 2.1: Obtención de Información concreta desde un Dataframe (parte 1)

Dataframe en base a ciertos criterios

DataFrame de filas que y = 1

```
In [3]: # creas un df con el formato del df que teníamos
# en aquellos casos en que la columna "y" del df es = 1
# se lo asignamos a la variable: "df_uno"
df_uno = df[df["y"]==1]
df_uno
```

```
Out[3]:
```

	X	y
0	10	1
1	20	1
3	40	1

Figura 2.2: Obtención de Información concreta desde un Dataframe (parte 2)

```
In [4]: # NombreDataframe.NombreColumna (otra posible forma de "llamar a una columna")
# Lo hemos llamado _b para guardarlo con otro nombre de variable
# es preferible, en este caso, no obstante, la anterior elección
df_uno_b = df[df.y==1]
df_uno_b
```

```
Out[4]:
```

	X	y
0	10	1
1	20	1
3	40	1

Figura 2.3: Obtención de Información concreta desde un Dataframe (parte 3)

DataFrame de filas que y = 0

```
In [5]: # creas un df con el formato del df que teníamos
# en aquellos casos en que la columna "y" del df es = 0
# se lo asignamos a la variable: "df_cero"
df_cero = df[df["y"]==0]
df_cero
```

Out[5]:

	x	y
2	30	0
4	50	0

```
In [6]: # NombreDataframe.NombreColumna (otra posible forma de "llamar a una columna")
# lo hemos llamado _b para guardarlo con otro nombre de variable
# es preferible, en este caso, no obstante, la anterior elección
df_cero_b = df[df.y==0]
df_cero_b
```

Out[6]:

	x	y
2	30	0
4	50	0

Figura 2.4: Obtención de Información concreta desde un Dataframe (parte 4)

3. GRÁFICOS RÁPIDOS CON PANDAS PARA DATAFRAMES

Cuando trabajamos con datasets, en algunas ocasiones, no es necesario conocer las mejores librerías para Data Science.

Lo que necesitamos es plotear 4 gráficas rápidamente para poder obtener información rápidamente del dataframe.

Vamos a hacer algunos ejemplos donde pandas es más que suficiente.

Con el siguiente ejemplo que corresponde a “pantallazos” hechos sobre el propio Jupyter Notebook.

PRINCIPALES DEPENDENCIAS

```
In [1]: import pandas as pd  
import matplotlib.pyplot as plt
```

Creo un DataFrame para el ejemplo

```
In [2]: df = pd.DataFrame({"X": [10,20,30,40,50], "y": [15,5,10,8,6]})  
df.head()
```

Out[2]:

	X	y
0	10	15
1	20	5
2	30	10
3	40	8
4	50	6

Figura 3.1: Gráficos rápidos con pandas (parte 1)

Gráficos rápidos con pandas

.plot()

```
In [3]: df.plot()
```

```
Out[3]: <AxesSubplot:>
```

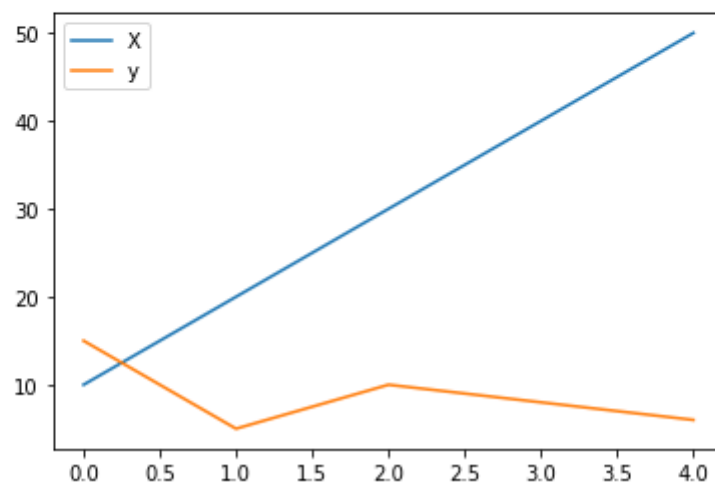


Figura 3.2: Gráficos rápidos con pandas (parte 2)

.plot() y plt.show()

```
In [4]: df.plot()  
plt.show()
```

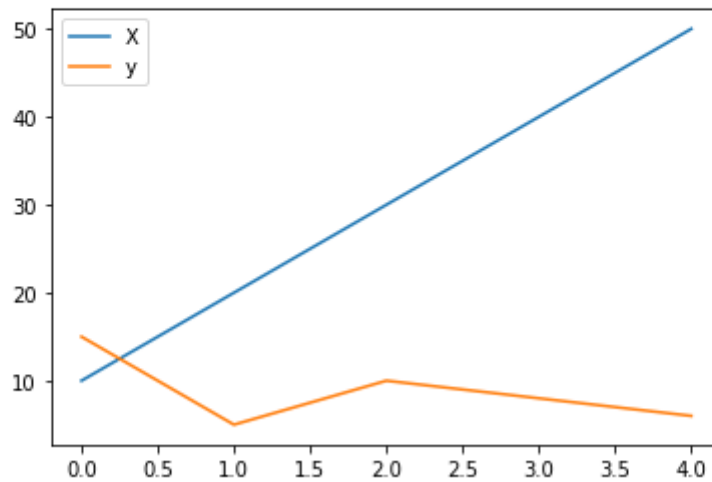


Figura 3.3: Gráficos rápidos con pandas (parte 3)

Gráfico de barras

```
In [5]: df.plot(kind="bar")  
plt.show()
```

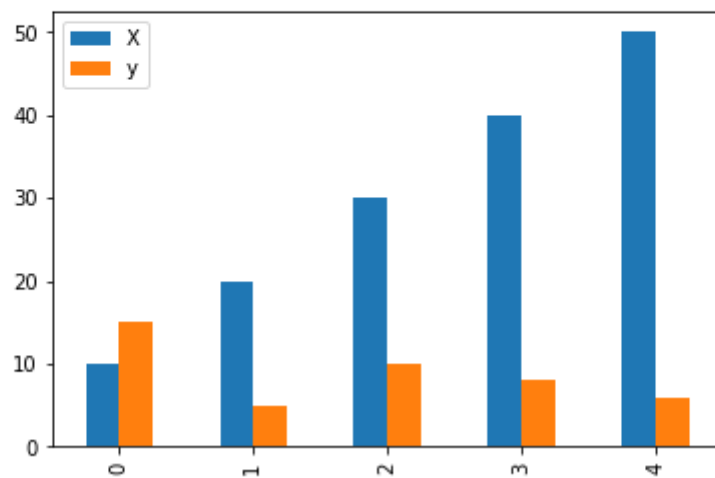


Figura 3.4: Gráficos rápidos con pandas (parte 4)

Stacked = apilado

```
In [6]: df.plot(kind="bar", stacked=True)
plt.show()
```

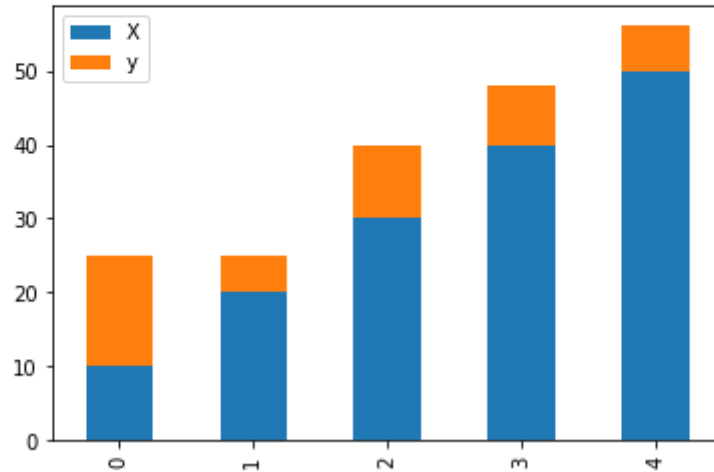


Figura 3.5: Gráficos rápidos con pandas (parte 5)

Buscamos más parámetros de df.plot

```
In [7]: df.plot?
```

```
Signature: df.plot(*args, **kwargs)
Type: PlotAccessor
String form: <pandas.plotting._core.PlotAccessor object at 0x000001AB4EC69C10>
File: c:\users\manut\appdata\local\programs\python\python38\lib\site-packages\pandas\plotting\_core.py
Docstring:
Make plots of Series or DataFrame.

Uses the backend specified by the
option ``plotting.backend``. By default, matplotlib is used.

Parameters
-----
data : Series or DataFrame
    The object for which the method is called.
x : label or position, default None
    Only used if data is a DataFrame.
y : label, position or list of label, positions, default None
    Allows plotting of one column versus another. Only used if data is a
    DataFrame.
```

Figura 3.6: Gráficos rápidos con pandas (parte 6)

Le pongo título y rejilla para verlo mejor

```
In [8]: df.plot(kind="bar", stacked=True,  
          grid=True, title="Gráfica ejemplo con pandas")  
  
plt.show()
```

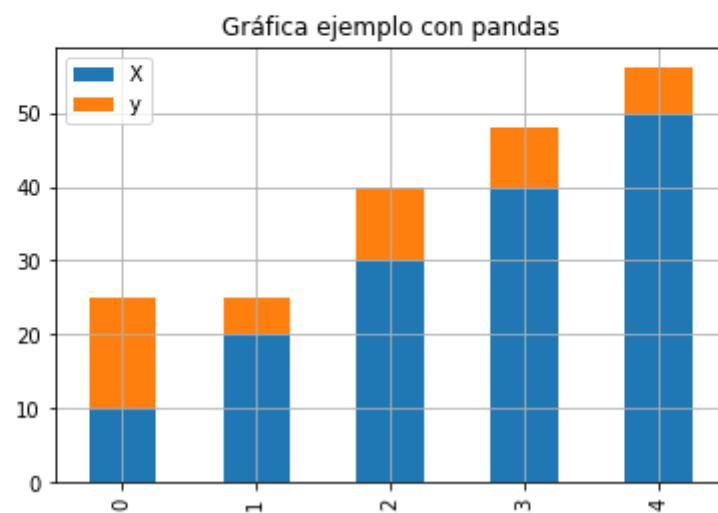


Figura 3.7: Gráficos rápidos con pandas (parte 7)

value_counts()

```
In [9]: df.y.value_counts()
```

```
Out[9]: 8      1  
        10     1  
        5      1  
        6      1  
        15     1  
        Name: y, dtype: int64
```

Símplemente añadimos .plot() y obtenemos la gráfica

```
In [10]: df.y.value_counts().plot(kind="bar")
```

```
Out[10]: <AxesSubplot:>
```

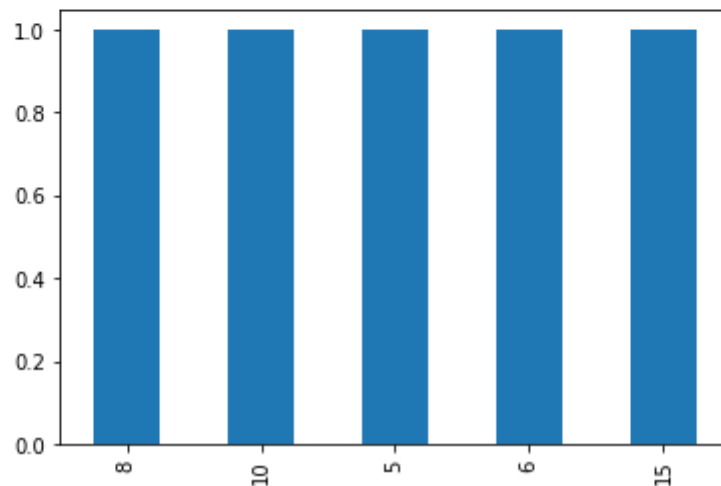


Figura 3.8: Gráficos rápidos con pandas (parte 8)

Pero, que si el dataset fuera diferente se vería mejor.

Haremos el siguiente ejemplo:

Imaginamos otro df

```
In [11]: df_2 = pd.DataFrame({"X": [10,20,30,40,50,60,70,80],  
                             "y": [10,20,10,40,20,30,20,10]})  
df_2
```

Out[11]:

	X	y
0	10	10
1	20	20
2	30	10
3	40	40
4	50	20
5	60	30
6	70	20
7	80	10

Figura 3.9: Gráficos rápidos con pandas (parte 9)

```
In [12]: df_2.y.value_counts()
```

```
Out[12]: 10    3  
        20    3  
        40    1  
        30    1  
        Name: y, dtype: int64
```

```
In [13]: df_2.y.value_counts().plot(kind="bar")  
plt.show()
```

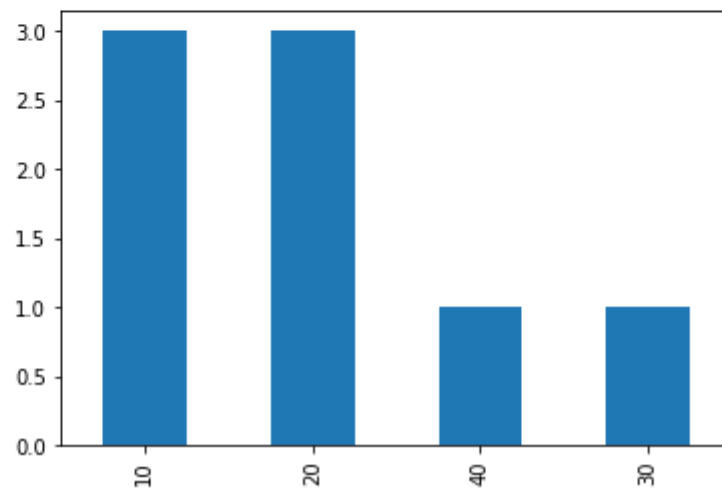


Figura 3.10: Gráficos rápidos con pandas (parte 10)

Con color diferente..

```
In [14]: df_2.y.value_counts().plot(kind="bar",  
                                         color=["green", "orange"])  
plt.show()
```

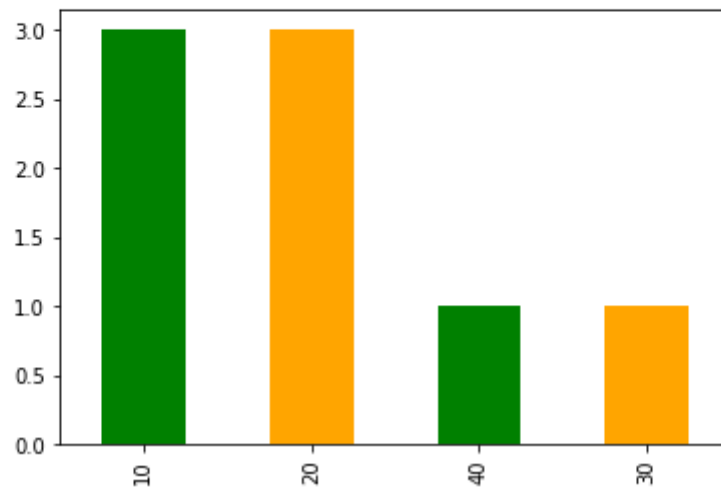


Figura 3.11: Gráficos rápidos con pandas (parte 11)

O incluso..

```
In [15]: df_2.y.value_counts().plot(kind="bar",  
                                         color=["blue", "green", "red", "orange"])  
plt.show()
```

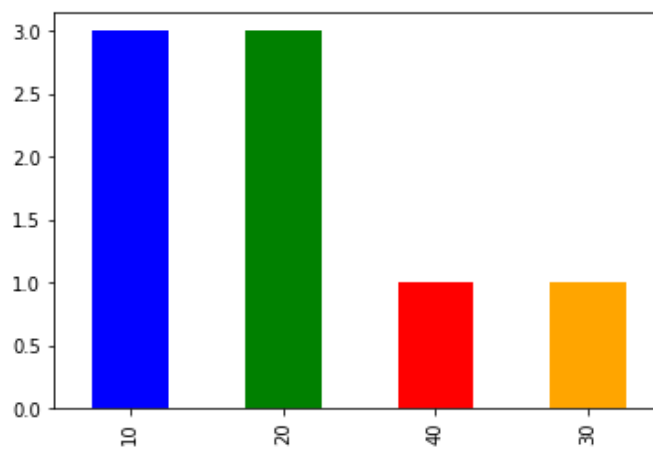


Figura 3.12: Gráficos rápidos con pandas (parte 12)

4. GRÁFICAS DEL IRIS DATASET – PARA EL PÉTALO

Recordamos que lo primero que se hace es importar las dependencias de nuestro proyecto, idealmente en las primeras celdas de Jupyter.

En mi caso tengo el .csv del Iris Dataset en una carpeta concreta por lo cual necesito indicarle la ruta para que pandas sea capaz de verlo.

Recordar también que `df.head()` lo que me hace es imprimir las primeras 5 líneas para ver de una pasada rápida el contenido que tenemos.

Y, que podríamos hacer lo mismo con `.tail()` para las últimas 5 líneas.

PRINCIPALES DEPENDENCIAS

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

DATAFRAME

```
In [2]: df = pd.read_csv("C:/Users/Manut/ALL/datasets/DATA/2_IrisSpecies.csv")
df.head()
```

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Figura 4.1: Gráficas en Iris Dataset – para pétalo (parte 1)

GRAFICO CON PANDAS - PÉTALO

```
In [3]: #Grafico Sepalo - Longitud vs Ancho
figura = df[df.Species == 'Iris-setosa'].plot(kind='scatter',
                                             x='PetalLengthCm', y='PetalWidthCm',
                                             color='blue', label='Setosa',
                                             figsize=(6.4,6.4))
df[df.Species == 'Iris-versicolor'].plot(kind='scatter',
                                          x='PetalLengthCm', y='PetalWidthCm',
                                          color='orange', label='Versicolor',
                                          ax=figura)
df[df.Species == 'Iris-virginica'].plot(kind='scatter',
                                         x='PetalLengthCm', y='PetalWidthCm',
                                         color='green', label='Virginica',
                                         ax=figura)

figura.set_xlabel('Longitud')
figura.set_ylabel('Anchura')
figura.set_title('Gráfica para el Pétalo - con Pandas')
plt.show()
```

Figura 4.2: Gráficas en Iris Dataset – para pétalo (parte 2)

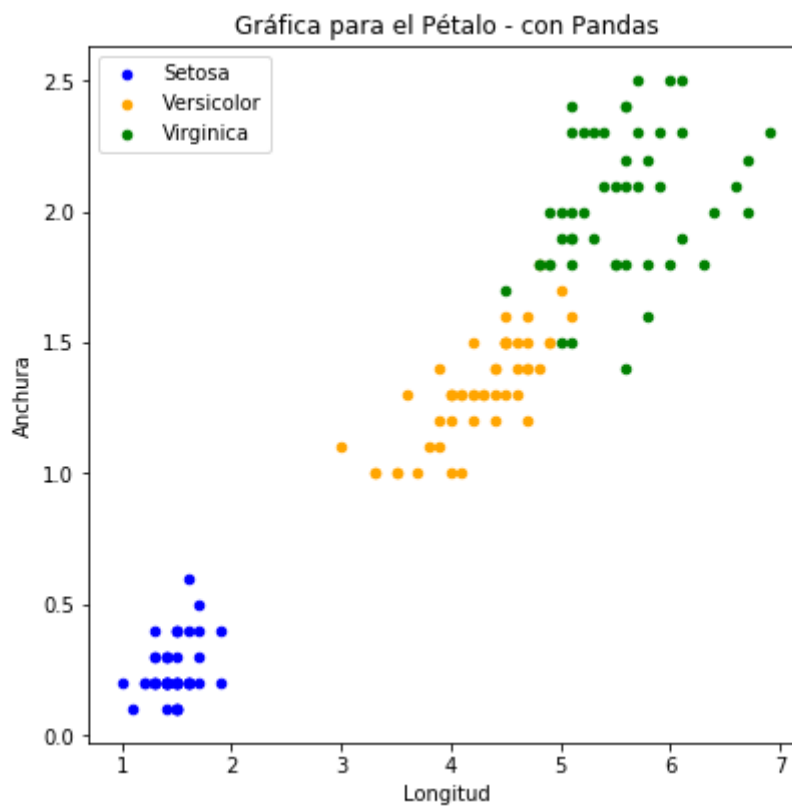


Figura 4.3: Gráficas en Iris Dataset – para pétalo (parte 3)

GRAFICO CON MATPLOTLIB- PÉTALO

```
In [4]: df_set = df[df.Species=="Iris-setosa"]
df_vers= df[df.Species=="Iris-versicolor"]
df_virg= df[df.Species=="Iris-virginica"]

fig = plt.figure(figsize=(6.4,6.4))
ax1 = fig.add_subplot(111)
ax1.scatter(df_set["PetalLengthCm"], df_set["PetalWidthCm"], s=10, c='blue', label='Setosa')
ax1.scatter(df_vers["PetalLengthCm"], df_vers["PetalWidthCm"], s=10, c='orange', label='Versicolor')
ax1.scatter(df_virg["PetalLengthCm"], df_virg["PetalWidthCm"], s=10, c='green', label='Virgínica')
plt.legend(loc='upper left');
plt.title("Gráfica para el pétalo - con Matplotlib")
plt.xlabel("Longitud")
plt.ylabel("Anchura")
plt.show()
```

Figura 4.4: Gráficas en Iris Dataset – para pétalo (parte 4)

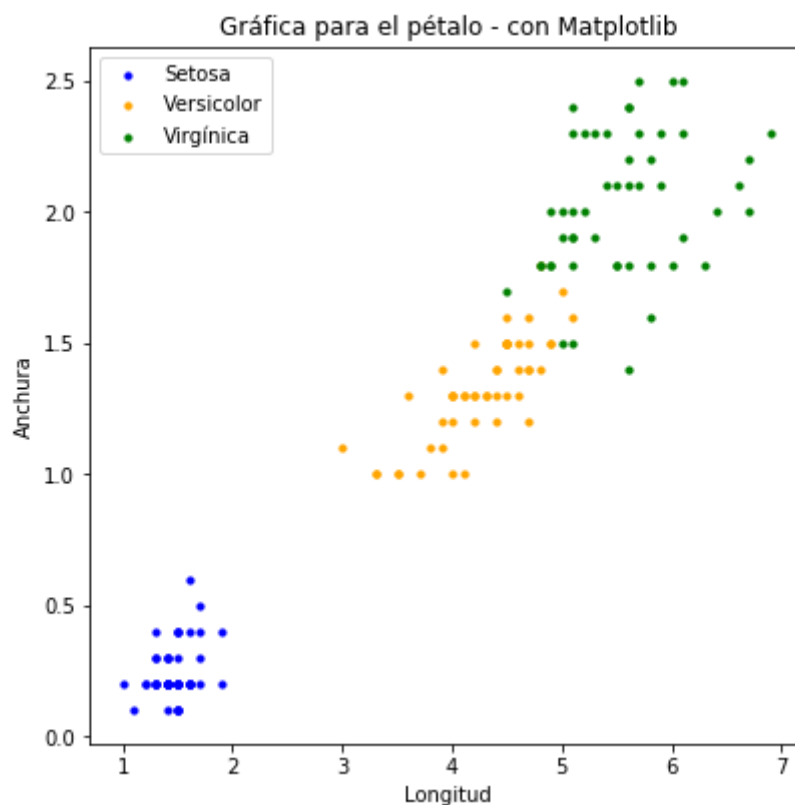


Figura 4.5: Gráficas en Iris Dataset – para pétalo (parte 5)

GRÁFICO CON SEABORN - PÉTALO

```
In [5]: # https://seaborn.pydata.org/installing.html
# !pip install seaborn

# Para el pétalo
sns.FacetGrid(df, hue="Species", height=6.4) \
    .map(plt.scatter, "PetalLengthCm", "PetalWidthCm") \
    .add_legend()

plt.title("Gráfica para el pétalo - con Seaborn")
plt.show()
```

Figura 4.6: Gráficas en Iris Dataset – para pétalo (parte 6)

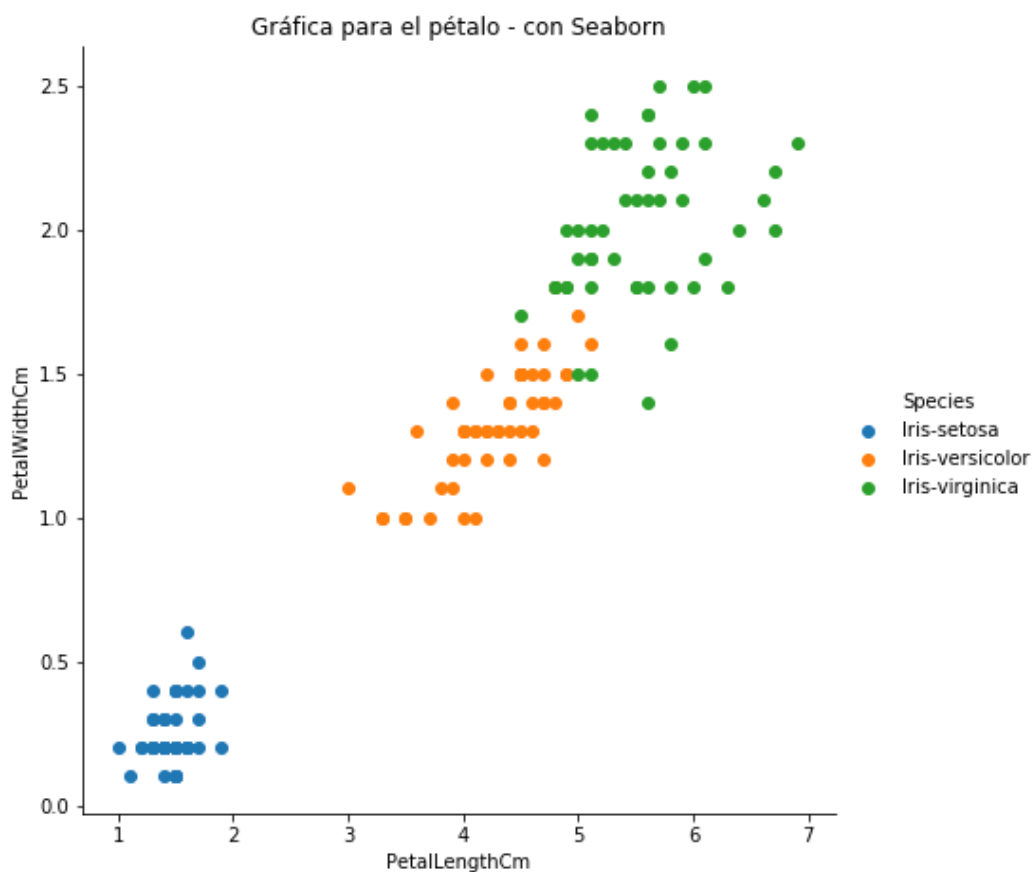


Figura 4.7: Gráficas en Iris Dataset – para pétalo (parte 7)

5. GRÁFICAS DEL IRIS DATASET – PARA EL SÉPALO

GRÁFICO PANDAS - SÉPALO

```
In [6]: #Grafico Sepalo - Longitud vs Ancho
figura = df[df.Species == 'Iris-setosa'].plot(kind='scatter',
                                             x='SepalLengthCm', y='SepalWidthCm',
                                             color='blue', label='Setosa',
                                             figsize=(6.4,6.4))

df[df.Species == 'Iris-versicolor'].plot(kind='scatter',
                                           x='SepalLengthCm', y='SepalWidthCm',
                                           color='orange', label='Versicolor',
                                           ax=figura)

df[df.Species == 'Iris-virginica'].plot(kind='scatter',
                                         x='SepalLengthCm', y='SepalWidthCm',
                                         color='green', label='Virginica',
                                         ax=figura)

figura.set_xlabel('Longitud')
figura.set_ylabel('Anchura')
figura.set_title('Gráfica para el Sépalo - con Pandas')
plt.show()
```

Figura 5.1: Gráficas en Iris Dataset – para sépalo (parte 1)

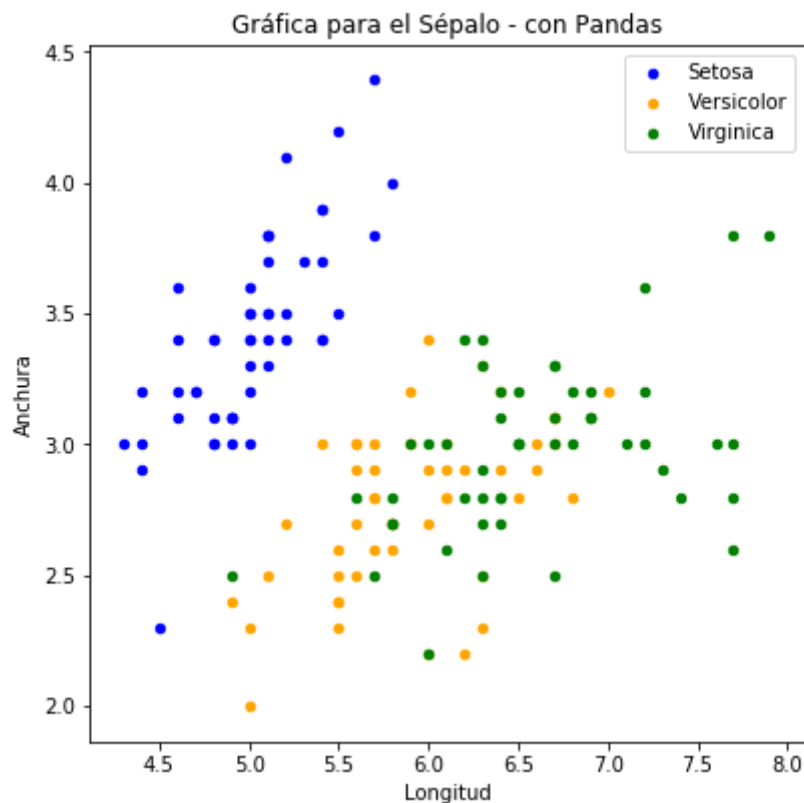


Figura 5.2: Gráficas en Iris Dataset – para sépalo (parte 2)

GRÁFICO MATPLOTLIB - SÉPALO

```
In [7]: df_set = df[df.Species=="Iris-setosa"]
df_vers = df[df.Species=="Iris-versicolor"]
df_virg = df[df.Species=="Iris-virginica"]

fig = plt.figure(figsize=(6.4,6.4))
ax1 = fig.add_subplot(111)
ax1.scatter(df_set["SepalLengthCm"], df_set["SepalWidthCm"], s=10, c='blue', label='Setosa')
ax1.scatter(df_vers["SepalLengthCm"], df_vers["SepalWidthCm"], s=10, c='orange', label='Versicolor')
ax1.scatter(df_virg["SepalLengthCm"], df_virg["SepalWidthCm"], s=10, c='green', label='Virginica')
plt.legend(loc='upper right')
plt.title("Gráfica para el Sépalo - con Matplotlib")
plt.xlabel("Longitud")
plt.ylabel("Anchura")
plt.show()
```

Figura 5.3: Gráficas en Iris Dataset – para sépalo (parte 3)

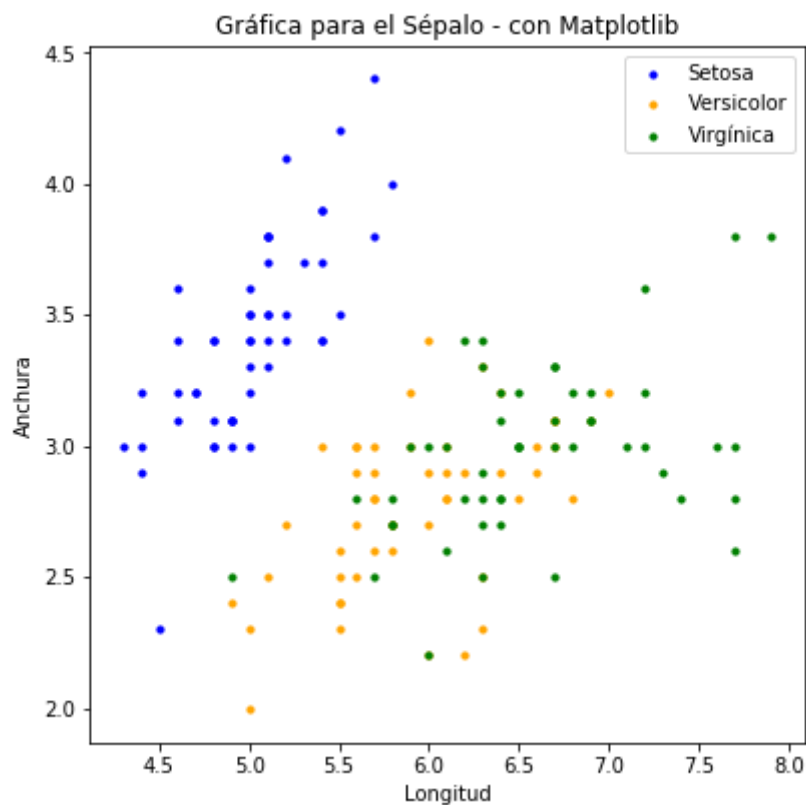


Figura 5.4: Gráficas en Iris Dataset – para sépalo (parte 4)

GRÁFICO SEABORN - SÉPALO

```
In [8]: # para el sépalo
sns.FacetGrid(df, hue="Species", height = 6.4) \
    .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
    .add_legend() \
    .set(title="Gráfica para el sépalo - con Seaborn")

# plt.title("Species en función del largo y ancho del sépalo")
plt.show()
```

Figura 5.5: Gráficas en Iris Dataset – para sépalo (parte 5)

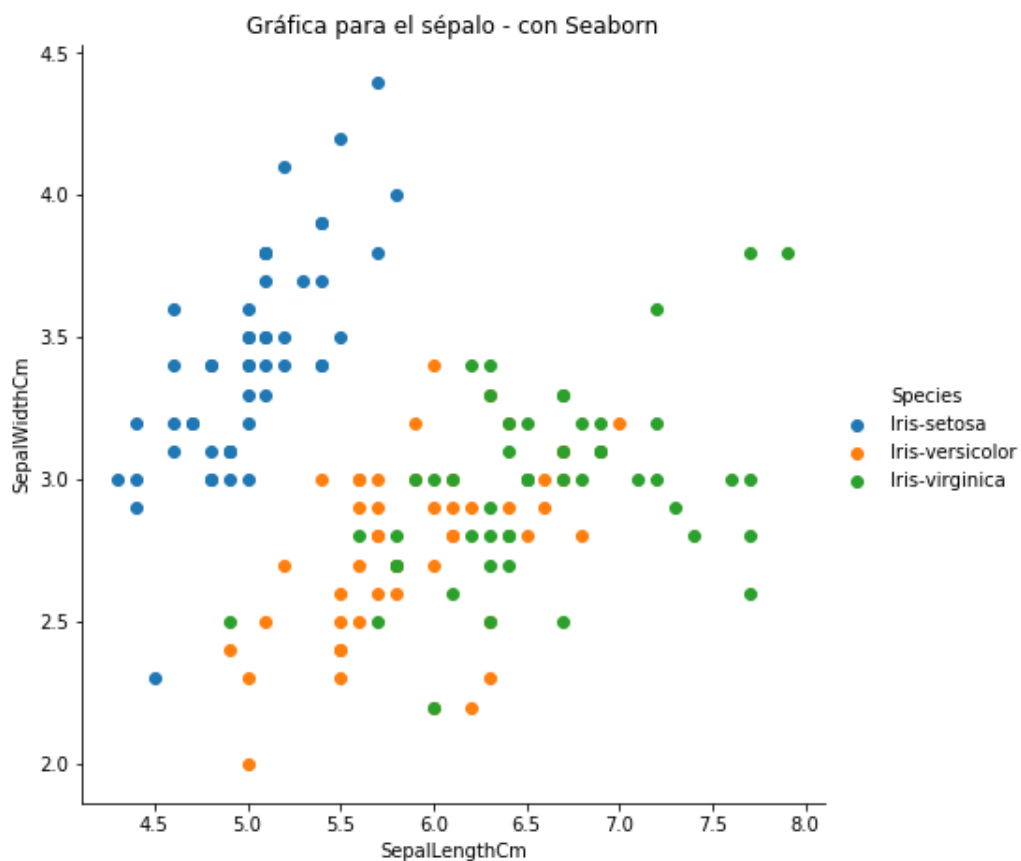


Figura 5.6: Gráficas en Iris Dataset – para sépalo (parte 6)

6. OTRAS LIBRERÍAS PARA HACER GRÁFICOS

Primeramente comentar que hay muchísimas.

Por comentar algunas:

ggplot:

<https://pypi.org/project/ggplot/>

<https://pypi.org/project/plotnine/>

Plotly, ya vista en Creación de Aplicaciones

Etc.

A día de hoy, Mayo-Junio 2021, se puede afirmar que en los próximos meses saldrán probablemente nuevas herramientas que nos permitan crear gráficos más visuales, e incluso, mejoras a las herramientas ya existentes.

El objetivo cuando se hacen gráficas no es necesariamente ofrecer gráficas de gran calidad, algunas veces, con pandas es más que suficiente para hacer una gráfica que nos permita ver cierta información de nuestro DataFrame y, con ello tomar conclusiones.

7. HERRAMIENTAS AVANZADAS PARA HACER GRÁFICOS

Tenemos muchas muy buenas opciones. No comentaremos todas ellas, pero sí algunas.

Primero es importante decir que algunas de ellas no requieren de Python para hacer gráficas de gran calidad, otras, en cambio, sí.

Por mencionar algunos ejemplos comentaremos:

- Dash: Ya vista en Aplicaciones Python
- Bokeh: Mencionada en Aplicaciones Python aunque no usada
- Power BI de Microsoft, de las más demandadas a nivel laboral
- Tableau (pronunciado “Tabló”) una de las mejores
- Savvy: Usada en empresas de gran nivel en EEUU
- Folium: Para mapas

A la hora de utilizar una u otra herramienta nos fijaremos en la necesidad.

No es lo mismo que:

- queramos hacer una pequeña visualización de los datos para entender qué tenemos y sacar 4 conclusiones,
- queramos presentar un informe en la empresa
- queramos presentar un importante informe a un cliente

Dependerá también de si queremos hacer algo:

- Sin necesidad de programar
- Programando nuestra aplicación

8. HERRAMIENTAS AVANZADAS 1: DASH

Pudimos ver en “Creación de Aplicaciones Python” esta gran herramienta.

Primeramente explicábamos conceptos de series temporales,

posteriormente, la forma de obtener datos de cotizaciones en bolsa con quandl y Yahoo Finance, y finalmente como crear una pequeña aplicación para mostrar las gráficas de cotización.

Dentro de esta herramienta sería interesante aprender algunas cosas más, pero como base puede ser suficiente.

Un ejemplo más avanzado sería hacer predicción de precios en el mercado de valores, tal y como se comentó con ARIMA, con Facebook Prophet, PyCaret, etc.

Puesto que la finalidad de esta asignatura es aprender a hacer gráficas y este Framework ya fue visto no profundizaremos más.

Tal vez tengamos ocasión de usarlo en alguna otra asignatura, ¿tal vez en la próxima asignatura de Big Data? O incluso en alguna de las asignaturas de Inteligencia Artificial.

En temas de Big Data, por cierto, es una gran elección.

9. HERRAMIENTAS AVANZADAS 2: BOKEH

Esta fue una herramienta mencionada en Creación de Aplicaciones Python.

Finalmente no se vieron ejemplos, dado que se explicó Dash, y se vieron otros muchos Frameworks.

(Frameworks GUI, Framework Full-Stack, Frameworks para Machine Learning, etc). Era realmente difícil añadir más cosas.

En el Framework Dash, y si nos acordamos de la clase que se impartió (aunque no de los apuntes del manual) es posible hacer Aplicaciones Dash en el propio Entorno Jupyter.

Y, si recordamos, había 2 opciones por lo menos, una de ellas, la opción en la cual se mostraba “embebida” la aplicación en el propio Jupyter Notebook, y otra opción en la cual se abría la aplicación fuera del propio Jupyter.

En el caso de Bokeh, también es posible usar Jupyter.

https://docs.bokeh.org/en/latest/docs/user_guide/jupyter.html

(en este link se ha dejado información acerca de ello).

Se pedirá algo (muy muy simple) relacionado con ello en la actividad de este tema donde se tratará de fomentar la proactividad de el/la alumno/a.

Pero no nos adelantemos, y sigamos viendo cosas.

10. HERRAMIENTAS AVANZADAS 3: POWER BI

A fecha de Junio 2021, momento en el que se confeccionan estos apuntes, se puede afirmar que es posible usar Power BI sobre Jupyter.

La noticia llegó hace muy muy poco y aprovecharemos la misma para incluirlo en este manual.

El link es el siguiente:

<https://powerbi.microsoft.com/en-us/blog/announcing-power-bi-in-jupyter-notebooks/>

Siendo una de las herramientas más demandadas en Data Science a nivel laboral, y teniendo integración con Jupyter, se hace atractivo probarlo.

11. HERRAMIENTAS AVANZADAS 4: TABLEAU

Una de las mejores opciones,

Consiste en un Software de pago, con una licencia Gratuita Tableau Public.

Para hacer gráficas de gran calidad podemos usar esta herramienta, sin la necesidad de programar.

El Link de Tableau es el siguiente:

<https://www.tableau.com/es-es>

El presente programa tiene su énfasis en Python, por lo cual no sé si tiene sentido explicar mucho acerca de Tableau (pronunciado “Tabló), pero muchos/as de vosotros/as supongo que trabajar como Data Scientist será un posible objetivo.

No quiere decir que sin saber Tableau no se pueda, pero si se cuenta con esta herramienta será más fácil.

Si es cierto que muchas veces, si se domina esta herramienta, la empresa para la que trabajas aproveche esas skills para que se trabaje con ella a tiempo completo como “Analista de datos” o “data analyst”.

Existe a su vez una certificación en Tableau, que pudiera ser interesante, para aquellas personas que dominen menos Python, y quizá quieran aprovechar esta “vía”,

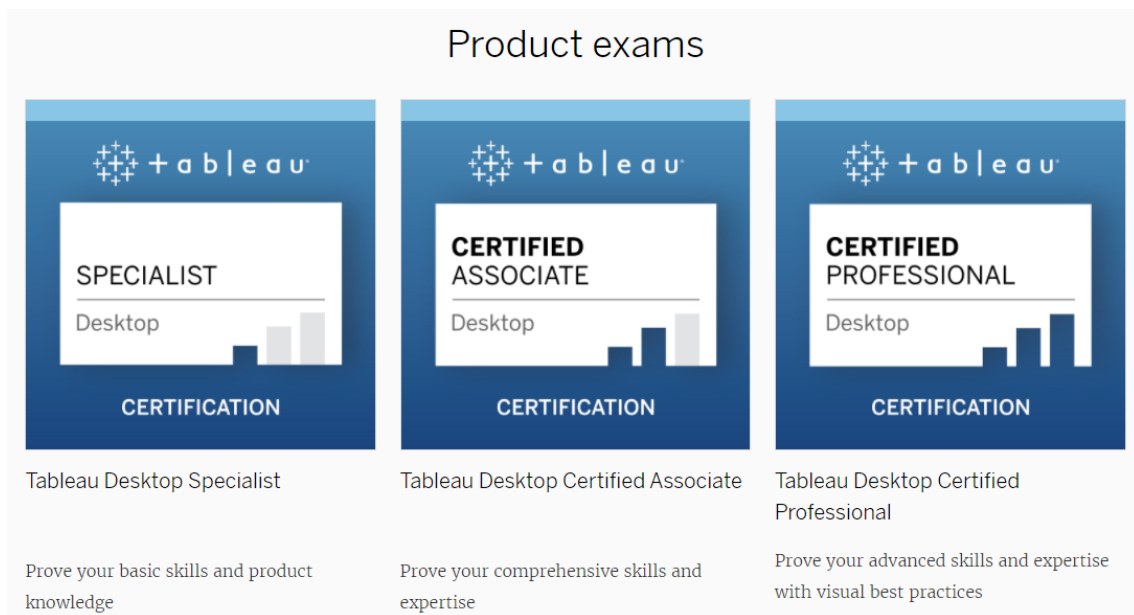


Figura 11.1: Algunas certificaciones posibles en Tableau

Que aunque existen algunas más. Son 3 de las posibilidades.

El Link para poder obtener más información es el siguiente:

https://www.tableau.com/learn/certification?utm_campaign_id=2018159&utm_campaign=Nurture-CUST-ALL-ALL-ALL-ALL&utm_medium=Paid+Search&utm_source=Google+Search&utm_language=EN&utm_country=Pan-EMEA&kw=tableau%20certification&adgroup=CTX-Brand-Certification-E&adused=ETA&matchtype=e&placement=&gclid=ds&gclid=ds

12. HERRAMIENTAS AVANZADAS 5: SAVVY

En el caso de la persona que preparó este contenido nunca trabajó con esta herramienta.

Pero tras ver que empresas como Facebook usan esta herramienta, y algunos de sus reclutadores en el perfil de LinkedIn afirman que aquellas personas que contratan “quieren ver esta herramienta en su perfil” me pareció interesante añadirla.

Obviamente, no es suficiente con saber Savvy, si quieres saber qué requisitos se piden en las más grandes Tecnológicas del planeta para trabajar en Data Science, te podría sugerir que hagas alguna pequeña búsqueda, pero ya se puede adelantar que, en general, las pruebas no son nada fáciles.

Por cierto, si viste “Python avanzado” seguro que te suena la parte de “Estructuras de datos” (listas, diccionarios etc), y se puede afirmar que en estas empresas (y en muchas otras) para trabajar como Data Scientist se pide una prueba llamada “Data Structures & Algorithms” (Estructuras de datos y Algoritmos) donde te podrían pedir que obtengas números Fibonacci menores de 1000, etc etc.

De modo que, no solo es saber Savvy, y no solo saber Big Data y Machine Learning. A veces se piden muchas más cosas, siendo, Savvy, el tema presente una de las cosas.

Nota: En Europa suele pedirse más Tableau y/o Power BI.

Link:

<https://savvy.readthedocs.io/en/latest/getting-started.html>

13. HERRAMIENTAS AVANZADAS 6: FOLIUM

Cuando trabajas con planos, una herramienta ideal puede ser Folium.

El Link donde obtener más información es el siguiente:

<https://python-visualization.github.io/folium/>

Getting Started

To create a base map, simply pass your starting coordinates to Folium:

```
[1]: import folium  
  
m = folium.Map(location=[45.5236, -122.6750])
```

To display it in a Jupyter notebook, simply ask for the object representation:

```
[2]: m
```

[2]:

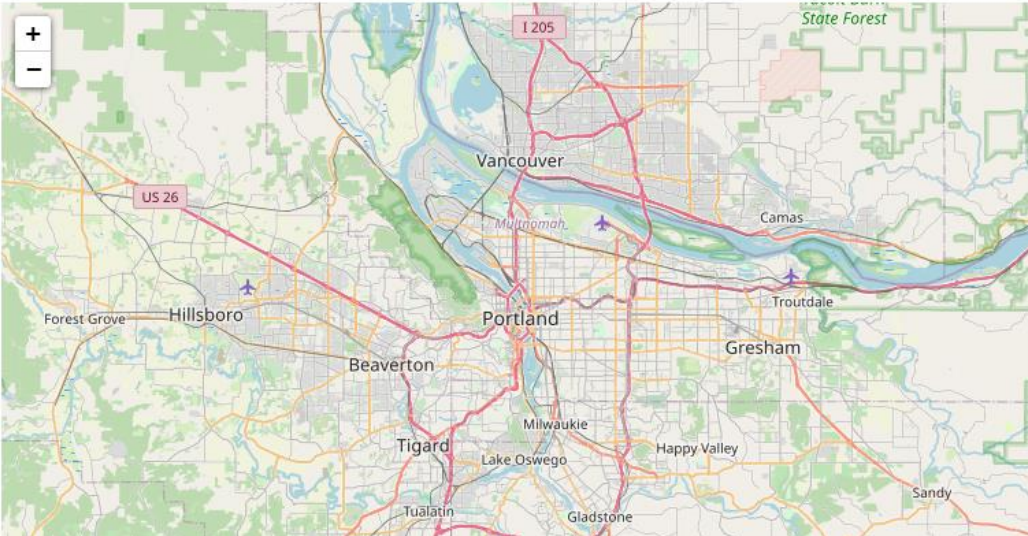


Figura 13.1: Getting Started (manual de aprendizaje para iniciantes de Folium)

14. PUNTOS CLAVE

- | Existen muchas opciones para hacer gráficas en Python.
- | Una opción fácil y rápida en algunos casos de plotear puede ser usar `.plot()` a pandas DataFrames.
- | En la práctica conocer pandas no es suficiente, dado que se necesitaría mucho tiempo para hacer gráficas de mayor calidad.
- | Conocer opciones avanzadas para hacer gráficas con Python puede ser de gran utilidad. Conocemos Dash de una asignatura previa, entonces ya disponemos de una herramienta importante en este sentido, aunque hay más.

